УДК 519.688

АЛГОРИТМЫ ОПТИМИЗАЦИИ ВЫЧИСЛЕНИЙ МЕТОДОМ ПСМ НА ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛИТЕЛЬНЫХ КЛАСТЕРАХ

П. В. Ващенков[1], **А. В. Кашковский**[1], **М. С. Иванов**[1]

Предложены алгоритмы динамической декомпозиции расчетной области при решении задач динамики разреженного газа методом ПСМ на параллельных вычислительных кластерах. Рассмотрена работа алгоритмов и их эффективность с использованием программного комплекса SMILE++. Работа выполнена в рамках следующих проектов: Междисциплинарный интеграционный проект СО РАН № 26, Программа фундаментальных исследований Президиума РАН № 11 и Междисциплинарный интеграционный проект СО РАН № 40.

Ключевые слова: ПСМ, параллельные вычисления, кластеры, алгоритмы, балансировка загрузки процессоров.

1. Метод прямого статистического моделирования. В методе прямого статистического моделирования (ПСМ) течение газа моделируется с помощью большого числа частиц, которые движутся и взаимодействуют между собой и поверхностью исследуемого объекта подобно молекулам реального газа. Метод ПСМ [1, 2] основан на расщеплении непрерывного движения и столкновения молекул на временно́м шаге на два последовательных этапа: свободномолекулярный перенос и столкновительную релаксацию. Фактически, в настоящее время этот метод стал основным инструментом для исследования сложных многомерных течений разреженного газа. Это обусловлено рядом его очевидных достоинств: сравнительной простотой перехода от одномерных к двух- и трехмерным задачам и возможностью использования различных моделей взаимодействия частиц газа, в том числе и моделей внутренних степеней свободы молекул и химических реакций, без значительного усложнения вычислительного алгоритма.

В процессе реализации метода ПСМ расчетная область разбивается на ячейки, размеры которых должны быть меньше локальной длины свободного пробега молекул. Величина временно́го шага Δt должна быть меньше среднего времени между столкновениями частиц. На временно́м шаге Δt в следующей последовательности рассматриваются межмолекулярные столкновения и свободное движение молекул.

1. В каждой ячейке проводятся столкновения частиц, принадлежащих только данной ячейке. Столкновения частиц из соседних ячеек не рассматриваются. Так как изменение функции распределения внутри ячейки считается малым, относительное расстояние между сталкивающимися частицами не учитывается. Скорости молекул после столкновения рассчитываются в соответствии с законами сохранения импульса и энергии.

2. Все частицы, находящиеся в расчетной области, передвигаются в соответствии со своими скоростями в текущий момент времени на расстояние $\Delta \boldsymbol{r} = \boldsymbol{v}\Delta t$. Если при этом частица вышла за пределы расчетной области, то слежение за ней прекращается. Если в процессе движения молекула попадает на тело, то моделируется ее отражение в соответствии с заданным законом взаимодействия газа с поверхностью и перевычисляется ее скорость после отражения. Аэродинамические характеристики тела вычисляются через импульс, переданный частицами телу при столкновении. На этом же шаге осуществляется генерация новых частиц, входящих в расчетную область, в соответствии с заданной на ее границах функцией распределения.

2. Параллелизация метода ПСМ. При параллелизации метода ПСМ наиболее часто используется разделение расчетной области. Ячейки, на которые разбита область, распределяются каким-либо образом между процессорами. Вся статистическая информация в ячейке, а также находящиеся в ней частицы хранятся на том же процессоре, которому принадлежит ячейка. Столкновение частиц в ячейке выполняется каждым процессором независимо от других процессоров. Перенос частиц осуществляет тот процессор, на котором они находятся. При переносе частиц часть из них вылетает за пределы ячейки и

[1] Институт теоретической и прикладной механики им. С. А. Христиановича СО РАН, ул. Институтская, д. 4/1, 630090, г. Новосибирск; П. В. Ващенков, мл. науч. сотр., e-mail: vashen@itam.nsc.ru; А. В. Кашковский, науч. сотр., e-mail: sasa@itam.nsc.ru; М. С. Иванов, зав. лаб., e-mail: ivanov@itam.nsc.ru

может оказаться в ячейке, которая принадлежит другому процессору. Такие частицы пересылаются на другой процессор. Очевидно, что для выполнения столкновений на следующем шаге процессор должен получить все частицы, которые пересылаются ему с других процессоров. Поэтому после этапа переноса и пересылки частиц происходит синхронизация, когда процессоры дожидаются выполнения вычислений всеми другими процессорами, и, убедившись, что вся необходимая информация отправлена и получена, переходят к выполнению следующего шага.

При параллелизации метода ПСМ следует учитывать некоторые особенности.

1. Работа процессора на проведение столкновений между частицами в ячейке зависит от количества частиц в этой ячейке. Чем их больше, тем больше времени нужно на моделирование столкновений. Так как число частиц в ячейке зависит от структуры течения (в области ударных волн оно выше, в следе за телом ниже), то число частиц в различных ячейках может отличаться в сотни раз.

2. Метод ПСМ начинает вычисления с равномерного потока, в котором в процессе счета формируются ударные волны, зоны повышения и понижения плотности. По завершении формирования структуры течения решение выходит на стационарный режим, при котором выполняется накопление статистической информации. Нестационарный участок, на котором происходит формирование поля течения, как правило, довольно длительный и требует выполнения нескольких тысяч шагов. На этом этапе число частиц в ячейке отслеживает динамику изменения течения и значительно изменяется. Соответственно, изменяется работа, выполняемая процессором для данной ячейки, и изменяется загрузка процессора.

3. В установившемся стационарном состоянии, когда средние параметры течения не изменяются во всем поле, число частиц в ячейке постоянно флуктуирует (становится выше или ниже среднего значения). Максимальное отклонение числа частиц пропорционально $\sqrt{\overline{n}}$, где $\overline{n}$ — среднее число частиц в ячейке. Поэтому работа процессора для одной и той же ячейки для разных шагов может существенно отличаться.

4. За исключением каких-либо очень специфических случаев, частица, передвигаясь по ячейке, вылетит из нее и попадет в другую ячейку (или покинет расчетную область). Поскольку число частиц в области очень большое (от сотен тысяч до миллионов), число частиц, вылетающих из ячеек на каждом шаге, тоже достаточно велико. Так как значительная часть из них может попасть на другой процессор, то и межпроцессорный обмен на каждом шаге может оказаться достаточно большим — до нескольких мегабайт.

3. Оценка эффективности параллелизации метода ПСМ. В идеальном случае время вычислений на параллельных компьютерах уменьшается прямо пропорционально увеличению числа процессоров. Однако в силу ряда причин (неравномерность распределения вычислений между процессорами, необходимость межпроцессорного обмена данными) вычисления выполняются медленнее, чем в идеальном случае. Отношение идеального времени вычислений к реальному называется “эффективностью параллелизации” и является критерием выбора методов и алгоритмов параллелизации.

Предположим, что методом ПСМ решается задача в области, состоящей из N ячеек на M процессорах. Ячейки необходимо распределить между процессорами. Такое распределение можно представить в виде матрицы m размерностью N столбцов и M строк. Если значение элемента матрицы $m(i,j)$ равно 1, то это означает, что ячейка j принадлежит процессору i. Поскольку ячейка не может находиться на нескольких процессорах, но обязательно должна быть на каком-либо из них, в матрице m в каждом столбце должна быть одна и только одна единица, а все остальные значения в этом столбце — нули.

Пусть q_j — время, затраченное процессором для вычислений в ячейке j. Тогда $Q_i = \sum\limits_{j=1}^{N} m(i,j)q_j$ — время вычислений для всех ячеек, находящихся на этом процессоре.

Суммарное время выполнения всей работы на всех процессорах $Q = \sum\limits_{i=1}^{M} Q_i = \sum\limits_{j=1}^{N} q_j$ является величиной постоянной и не зависящей от числа процессоров.

Если время вычислений Q_i на разных процессорах разное, то из-за синхронизации процессоры будут дожидаться окончания работы самого медленного (наиболее загруженного) процессора, а время его выполнения вычислений на шаге фактически будет временем выполнения шага. Очевидно, что если все процессоры загружены одинаково, то они не будут ждать друг друга; тогда время выполнения шага будет минимальным и равным $\overline{Q} = \dfrac{Q}{M}$. Эффективность загрузки процессоров вычисляется по формуле $e_l = \overline{Q} / \max\limits_{i=1}^{M} Q_i$.

Пусть τ_{jk} — время, затраченное на передачу информации из ячейки j в ячейку k, если они находятся на разных процессорах. Если ячейки находятся на одном и том же процессоре, то пересылка не произво-

дится, и время передачи равно нулю. Время передачи данных с процессора i на все другие процессоры, таким образом, выражается формулой

$$\Theta_i = \sum_{j=1}^{N} \sum_{k=1}^{N} m(i,j)\big(1 - m(i,k)\big)\tau_{jk}, \tag{1}$$

где множитель $m(i,j)\big(1-m(i,k)\big)$ равен единице только в том случае, если происходит пересылка с данного процессора в ячейку, принадлежащую другому процессору, а в остальных случаях равен нулю. Так как пересылка осуществляется процессором после выполнения всей работы, за эффективность параллелизации можно принять выражение $e = \dfrac{\overline{Q}}{\max\limits_{i=1}^{M}(Q_i + \Theta_i)}$.

Таким образом, для эффективных вычислений методом ПСМ необходимо подобрать такую матрицу m, при которой загрузка на каждом процессоре близка к средней $(\overline{Q})$, а время передачи (Θ_i) минимальное. Выбор матрицы m — неоднозначная задача. Более того, в силу перечисленных особенностей метода ПСМ, для каждого шага существует своя оптимальная матрица, получить которую до начала выполнения шага невозможно, потому что величины q_j и τ_j еще не известны. Именно поэтому разрабатываются различные алгоритмы, которые с учетом усредненных значений q_j и τ_{jk} вычисляют достаточно оптимальную матрицу m для большого числа шагов.

4. Критерии оценки загрузки процессора. Поиск оптимальной матрицы m базируется на знании средних величин времени выполнения работы в ячейке q_j и времени пересылки на другой процессор τ_{jk}.

На практике получить значения этих времен почти невозможно. Поэтому обычно замеряют время начала и окончания выполнения какой-либо операции: их разность дает время выполнения операции. Однако таймер, т.е. функция, которая получает текущее время от процессора, обращается к ядру операционной системы и сам выполняется определенное время. Более того, поскольку ядро операционной системы квантует выполнение задач, то вызов таймера может давать неверное значение времени из-за того, что в данный момент выполнение задания было временно прервано ядром. Так, на процессоре Intel(R) Pentium(R) 4, 2.80 GHz под управлением операционной системы Ubuntu 7.10 замеры выполнения стандартной C-функции times показали, что среднее время выполнения составляет 0.6–0.8 мкс, в то время как максимальное время срабатывания таймера было 0.01 с (на несколько порядков больше).

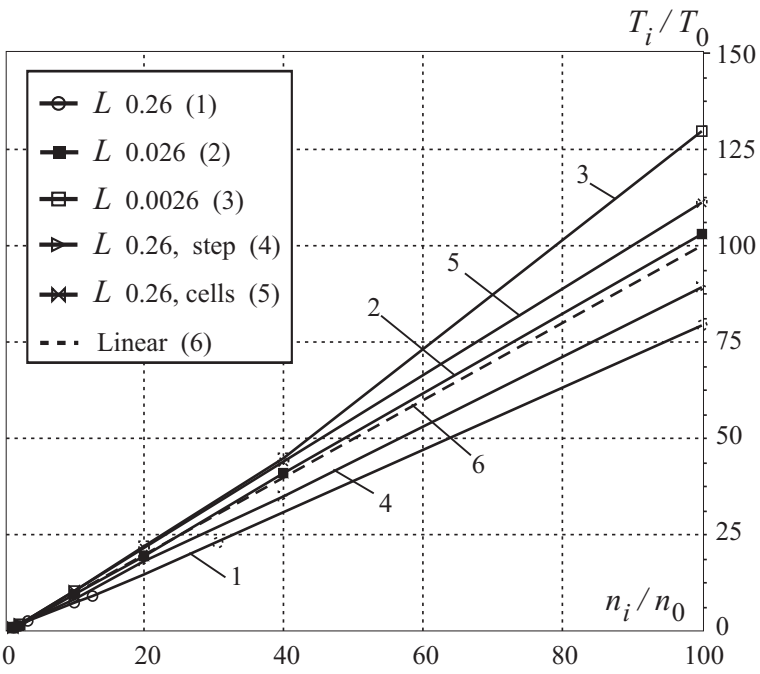

Рис. 1. Относительное время вычислений при изменении числа частиц в ячейке

Если попытаться засечь, за какое время процессор перемещает одну частицу (т.е. в большинстве случаев выполняет операцию $\Delta \boldsymbol{r} = \boldsymbol{v}\Delta t$), то время операции окажется меньше времени срабатывания таймера и тем более ошибки, которую он допускает. Хотя процесс моделирования столкновений всех частиц в ячейке более длительный, на него также наложатся ошибки в определении времени.

Поэтому необходимы какие-либо косвенные критерии определения времени выполнения вычислений для каждой ячейки, которые не требовали бы частого вызова таймера и больших дополнительных вычислений.

В качестве такого критерия предлагается использовать число частиц в ячейке. Как уже уже говорилось, с возрастанием числа частиц возрастает время вычислений, причем это возрастание практически линейно. Это легко показать на примере моделирования равномерного потока, в котором вариации числа частиц, размера ячеек и средней длины свободного пробега молекул не влияют на структуру течения,

а влияют только на время выполнения. Поэтому сначала были произведены вычисления для некоторого начального числа частиц в ячейке, а потом расчеты повторены для увеличенного в несколько раз числа частиц и для измененного числа ячеек при том же общем числе частиц (что также приводит к изменению среднего числа частиц в ячейке). Задание разных средних длин свободного пробега молекул (L) и вариация шага по времени изменяют число столкновений частиц в ячейке на каждом шаге и, соответственно, загрузку процессора.

Полученная зависимость возрастания относительного времени вычисления при относительном увеличении числа частиц в ячейке представлена на рис. 1. Приведены результаты для различных средних длин свободного пробега ($L = 0.0026, \ldots, 0.26$), измененного в 10 раз шага по времени (step) и измененного в 100 раз числа ячеек (cells). Пунктирной линией показана линейная (идеальная) зависимость. Как видно из графиков, время вычислений увеличивается практически пропорционально увеличению числа частиц, причем отклонение не превышает 25%.

Необходимо отметить, что использование числа частиц в ячейке n_{cj} вместо параметра q_j не позволяет получить абсолютное время выполнения работы процессора. Однако для вычисления критерия эффективности загрузки процессоров e_l достаточно иметь относительные значения

$$e_l \simeq e_l^\star = \frac{\overline{Q}^\star}{\max\limits_{i=1}^{M} Q_i^\star}, \tag{2}$$

где $e_l^\star$, $Q_i^\star$ и $\overline{Q}^\star$ — эффективность загрузки, затраты процессора на вычисления в ячейке i и средние временны́е затраты процессора на ячейку, посчитанные на основе числа частиц в ячейке:

$$Q_i^\star = \sum_{j=1}^{N} m(i,j) n_{cj}, \quad Q^\star = \sum_{i=1}^{M} Q_i^\star = \sum_{j=1}^{N} n_{cj}. \tag{3}$$

Критерием времени переноса частиц с одного процессора на другой вместо τ_{jk} может быть число $n_{\tau jk}$ пересылаемых частиц из ячейки j в ячейку k. Как показывает практика, при использовании библиотеки MPI в сети с пропускной способностью 100 Мбит/с потери времени на установление соединения и системное обслуживание составляют примерно 7%, т.е. практическая пропускная способность шины составляет примерно 93 Мбит/с. Зная размер передаваемой с каждой частицей информации достаточно точно, можно получить время передачи $\tau_{jk} = \dfrac{n_{\tau j,k} S}{0.93\, B}$, где S — размер пересылаемой с одной частицей информации в битах, а B — паспортная скорость передачи по каналу связи в битах в секунду. Для вычисления времени суммарной пересылки с процессора используется формула (1).

Поскольку вычисление $e_l^\star$ по формуле (2) ведется в относительных единицах, а τ_j — в абсолютных, по ним нельзя рассчитать эффективность параллелизации e. Поэтому необходимо рассматривать отдельно эффективность загрузки процессора $e_l^\star$ и эффективность переноса $e_\tau = \max\limits_{i=1}^{M} \Theta_i$, а наиболее эффективным будет считаться такой выбор матрицы m, при котором $e_l^\star$ близка к единице, а e_τ близка к нулю.

5. Эмулятор параллельных вычислений методом ПСМ. Использование критерия эффективности параллелизации, пропорционального числу частиц в ячейке, позволяет построить модель параллельных вычислений плотности и скорости методом ПСМ на основе полей течений. Действительно, среднее число частиц в ячейке может быть легко получено из плотности газа в ячейке, отнесенной к средней массе молекулы и объему ячейки, а перенос может быть представлен как поток частиц через грань ячейки. Обычно шаг по времени в ПСМ-методе выбирается так, чтобы частица могла пролететь расстояние, не превышающее размера ячейки. Поэтому для эмуляции параллельных вычислений можно считать, что перенос частиц осуществляется только в соседние ячейки, а в остальные — отсутствует. Таким образом, среднее число частиц n_{cj} в j-й ячейке и среднее число частиц $n_{\tau jk}$, пересылаемых в соседнюю ячейку, вычисляются по формулам $n_{cj} = \dfrac{\rho_j V_j}{\mu}$ и $n_{\tau jk} = n_{cj} v_{jk} S_{jk}$, где k — номер ячейки, соседней с ячейкой j, ρ_j и V_j — плотность и объем ячейки j, μ — масса молекулы, S_{jk} — площадь грани между ячейками j и k и v_{jk} — проекция скорости потока на нормаль к этой грани.

Поскольку оценка эффективности с использованием данной модели проводится по уже выполненным вычислениям без запуска новых параллельных вычислений, ожидания в очереди и т.д., то такая модель позволяет быстро протестировать вновь разрабатываемый алгоритм на большом числе различных задач. Это дает возможность оценить работоспособность разрабатываемых алгоритмов в приложении к разным видам течений газа и разному числу процессоров.

6. Алгоритмы разделения области по процессорам. Алгоритмы разделения (декомпозиции) области по процессорам могут быть статическими (когда в процессе вычислений разбиение области не меняется) и динамическими (когда в процессе вычислений периодически оценивается загрузка процессоров и производится перераспределение ячеек по процессорам).

7. Статический вероятностный алгоритм. Основная идея алгоритма заключается в том, что ячейки распределяются по процессорам случайным образом. Поэтому с большой вероятностью на каждый процессор попадают ячейки как из области плотного течения, так и из области разреженного течения. Такой алгоритм декомпозиции области дает очень равномерное распределение нагрузки по процессорам, причем равномерная загрузка сохраняется и на этапе установления, и на этапе накопления статистики. При использовании этого алгоритма декомпозиции неважно, изменяется ли течение или уже достигло стационарного состояния, поэтому и не требуется динамическое переразбиение области в процессе счета.

Данный алгоритм прост в реализации и для любой задачи дает заведомо хорошую балансировку загрузки процессоров. Он применяется в программном комплексе SMILE [3, 4]. Однако к его недостаткам следует отнести большой объем пересылаемых данных. Действительно, при использовании этого алгоритма высока вероятность, что две соседние ячейки не принадлежат одному процессору, т.е. на каждом временно́м шаге необходимо передавать другим процессорам практически все частицы. Для кластера с сетевым интерфейсом между процессорами этот факт может быть серьезным замедляющим фактором. Минимизация пересылки требует объединения соседних ячеек с учетом структуры течения, что приводит к использованию динамических алгоритмов.

8. Динамические алгоритмы. Алгоритмы, адаптируемые к решаемой задаче, должны выполняться в несколько этапов.

1. В течение нескольких шагов производится расчет с произвольным разделением области по процессорам. На этом этапе накапливается информация по числу частиц в ячейках и пересылкам.

2. На основе полученных данных расчетная область разбивается на подобласти таким образом, чтобы количество частиц в каждой подобласти было примерно равным, а суммарный поток частиц через границы подобластей был минимальным.

3. Через несколько расчетных шагов снова накапливается информация по числу частиц в ячейках и пересылкам и производится новое разделение области между процессорами.

Существует немало алгоритмов динамического разделения области для метода ПСМ (см. [5, 6] и др.). Однако все они, как правило, применялись на параллельных компьютерах с очень быстрым обменом, поэтому внимание уделялось только равномерности загрузки процессоров и не учитывалась пересылка данных. В рассматриваемых ниже динамических алгоритмах была сделана попытка минимизировать объем пересылаемой информации.

8.1. Алгоритм деления на равные части. Данный алгоритм является модификацией известного алгоритма деления пополам [5, 7]. Расчетная область делится на подобласти плоскостью, положение которой подбирается так, чтобы загрузка процессоров в обеих подобластях была одинаковой. Каждая из полученных подобластей также делится пополам и так далее, пока количество подобластей не станет равным числу процессоров. Следовательно, число процессоров должно быть равно 2^n, где n — целое число, что не всегда удобно.

Для учета межпроцессорного обмена предлагается вместо одной плоскости использовать три независимые плоскости, перпендикулярные координатным осям. Для каждой из них определяется положение, обеспечивающее одинаковую загрузку подобластей, и вычисляется объем передаваемой через плоскость информации. Для разделения на подобласти используется только одна из них, межпроцессорный обмен через которую минимальный. Подобласти делятся аналогично.

Каждую подобласть можно делить не на две части, а на три, пять или семь. Это позволяет использовать число процессоров, которое можно представить перемножением простых чисел. Например, 24 процессора можно представить в виде $2 \times 2 \times 2 \times 3$. Соответственно, сначала область три раза делится на две равные части, а потом каждая из подобластей делится на три.

Таким образом, в этом алгоритме сохраняется простота и скорость работы алгоритма половинного деления, но повышается универсальность и делается попытка уменьшить обмен данными между процессорами.

8.2. Алгоритм "Жизнь". Название алгоритма связано с тем, что его работа похожа на известную игру "Жизнь", моделирующую размножение колонии микроорганизмов. На начальном этапе работы алгоритма каким-либо образом из расчетной области выбираются ячейки по числу процессоров и назначаются каждая своему процессору. Эти ячейки будут являться центрами "жизни", и вокруг них будут наращиваться области ответственности процессоров.

Дальнейшая работа алгоритма состоит в поиске области с минимальной загрузкой, для которой выполняются следующие этапы.

а) Среди граничных ячеек области ищется ячейка с максимальным потоком частиц, который идет в никем еще не занятую ячейку или из нее. Определяется номер этой незанятой ячейки, и она присоединяется к области. Такое распространение области вдоль направления максимального потока частиц гарантирует, что максимальный обмен частицами будет происходить между ячейками, принадлежащими одному и тому же процессору.

б) Если область оказалась окружена другими областями и "свободных" ячеек по соседству нет, то новый центр "жизни" выбирается случайным образом из числа незанятых ячеек.

в) Цикл повторяется до тех пор, пока все ячейки не станут принадлежать своим процессорам.

Достоинство этого алгоритма в очень низкой величине дисбаланса загрузки вследствие того, что наращивание области происходит по ячейкам, т.е. очень небольшими порциями.

8.3. Алгоритм "Подвижные границы". В начальный момент времени область разбивается на равные по количеству ячеек подобласти вдоль какой-либо координатной оси. По формуле (3) вычисляется загрузка каждого процессора. Далее

а) ищется процессор, на котором выполняется максимальная работа;

б) среди ячеек, граничащих с подобластями других процессоров, ищется ячейка с максимальным потоком частиц через границу между процессорами;

в) эта ячейка передается другому процессору;

г) пересчитывается загрузка процессоров; процесс повторяется, пока дисбаланс загрузки не станет меньше заданной величины.

Как оказалось, эффективность алгоритма сильно зависит от первоначального разделения области. В настоящей статье рассматриваются два варианта начального разделения: полосами вдоль потока и поперек потока.

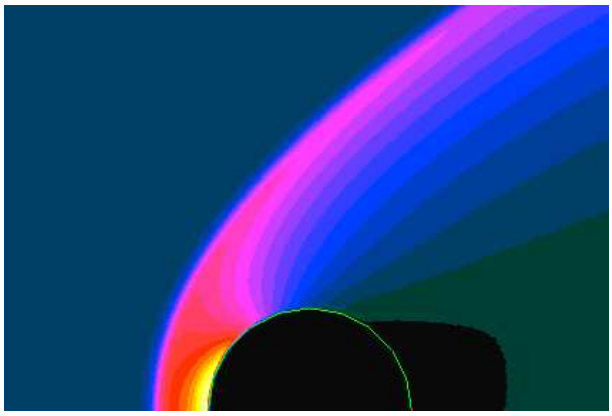

Рис. 2. Поле плотности около цилиндра: $\mathrm{Kn} = 0.01$, $\mathrm{M} = 5$

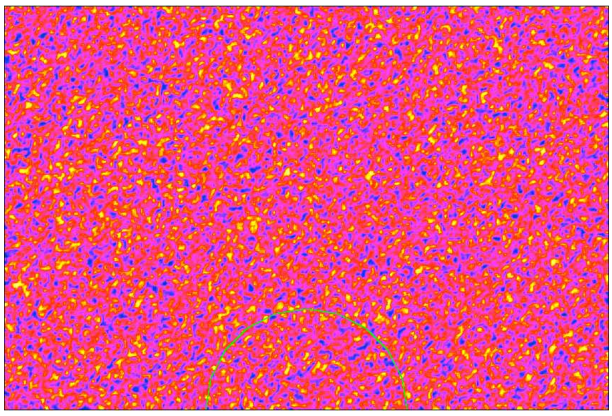

Рис. 3. Вероятностный алгоритм

9. Сравнение эффективности алгоритмов. Оценка эффективности представленных выше алгоритмов была сделана на примере расчета двухмерного поперечного обтекания цилиндра потоком газа при числе Кнудсена $\mathrm{Kn} = 0.01$ и скорости $\mathrm{M} = 5$ с использованием программного комплекса SMILE++ [8, 9]. На рис. 2 представлено поле плотности около цилиндра. Как можно видеть, значения плотности в разных областях пространства значительно различаются.

Для всех предлагаемых алгоритмов проводится предварительный этап расчета с произвольной декомпозицией области. Цель этого расчета — получить распределения плотности и скоростей в каждой ячейке. На основе этой информации каждый алгоритм по-своему разделяет область. Распределения ячеек на 24 процессорах при использовании вероятностного алгоритма и алгоритмов деления на равные части, "жизнь", подвижные границы с горизонтальной и вертикальной начальной разбивкой показаны соответственно на рис. 3–7. Области, принадлежащие одному процессору, окрашены одним цветом. Видно, что алгоритмы деления на равные части и "жизнь" "чувствуют" наличие ударной волны, в которой существенно изменяется загрузка и направление потоков. Поскольку основной поток частиц идет слева направо, использование горизонтальной начальной разбивки в алгоритме с подвижными границами более предпочтительно. Это хорошо видно на рис. 7, где алгоритм пытается вытянуть начальную вертикальную разбивку в горизонтальные полосы. Отчетливо видно, что все динамические алгоритмы "подстраива-

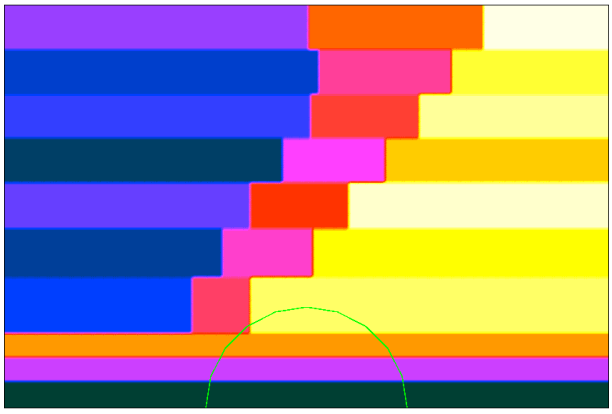

Рис. 4. Алгоритм деления на равные части

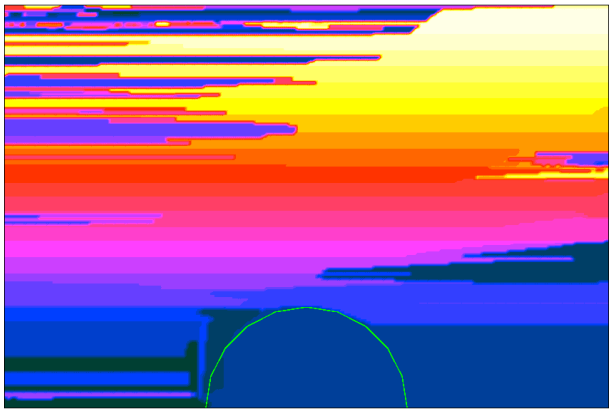

Рис. 5. Алгоритм “Жизнь”

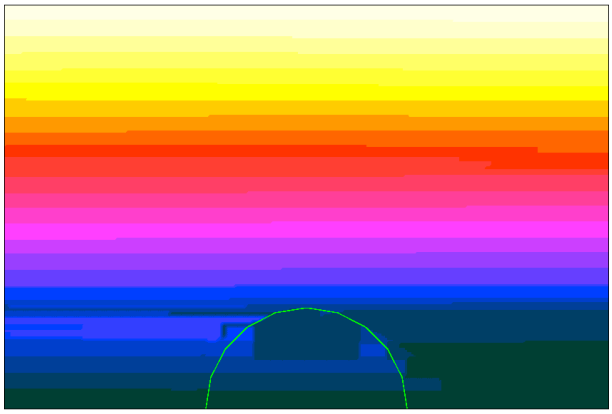

Рис. 6. Алгоритм “Подвижные границы”. Начальная разбивка вдоль потока

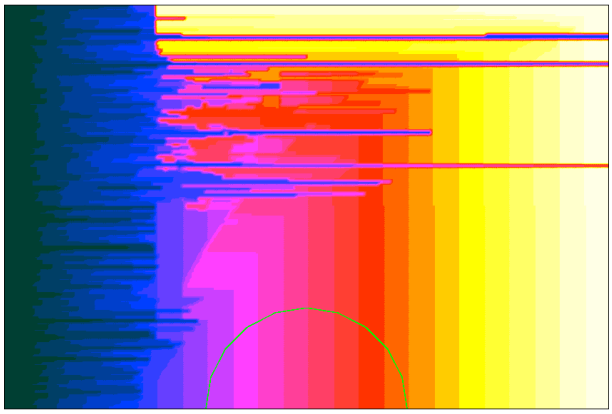

Рис. 7. Алгоритм “Подвижные границы”. Начальная разбивка поперек потока

ются” под течение газа и, в основном, объединение ячеек происходит вдоль линий тока. Эту особенность можно использовать при разработке других алгоритмов.

Вычисления проводились на кластере МВС-15000ВМ Объединенного суперкомпьютерного центра (Москва). Этот кластер оснащен 1070 процессорами PPC970FX и сетью Mirynet, 2 Гбит/с. Точно такие же расчеты проведены на кластере МВС-1000 Сибирского суперкомпьютерного центра СО РАН (Новосибирск), имеющем 32 процессора Alpha 21264 (833 МГц), объединенных сетью Mirynet.

Расчеты проводились на разном количестве процессоров: 1, 2, 4, 8, 16, 32 и 64 (кроме МВС-1000, на котором только 32 процессора). Эффективность параллелизации определялась по классической формуле $e = \dfrac{T_1/M}{T_M}$, где T_1/M — “идеальное” время расчета (время вычислений на одном процессоре, поделенное на число процессоров), а T_M — реальное время вычислений на M процессорах [7]. Полученная эффективность представлена в табл. 1.

Следует отметить, что значительное падение эффективности при увеличении числа процессоров связано с тем, что для оценки эффективности вычисления проводились на одном процессоре и параметры метода были выбраны так, чтобы задача поместилась в память одного узла и была посчитана в разумное время. Разделение той же самой задачи на большое число процессоров (в частности, 64) привело к тому, что процессоры были “недозагружены” — вычисления производились слишком быстро по сравнению с накладными расходами. В реальности для решения данной задачи не нужно было бы использовать больше восьми процессоров или можно было бы взять большее число частиц и ячеек, что улучшило бы пространственное разрешение и уменьшило статистическую ошибку.

Некоторые вычисления показали эффективность выше 1, чего теоретически быть не должно. Скорее всего, это связано с особенностью кэширования памяти процессора [7]. Если данные целиком помещаются в кэш процессора, то, поскольку доступ к нему существенно быстрее, чем к основной памяти, вычисления с такими данными проводятся быстрее и, кроме того, не требуется подкачки данных из основной памяти. Если при этом “накладные” расходы компьютера оказались низкими, то время вычислений может

оказаться меньше чем T_1/M.

Как и ожидалось, эффективность алгоритма "Подвижные границы" с начальной разбивкой поперек потока меньше (за исключением результата расчета на 8 процессорах), чем эффективность того же алгоритма с начальной разбивкой вдоль потока. Остальные алгоритмы дают примерно одинаковую эффективность. Очевидно, в реальной задаче желательно опробовать по очереди все алгоритмы и дальнейшие расчеты проводить с самым эффективным алгоритмом.

Таблица 1

Эффективность алгоритмов для разного числа процессоров

Алгоритм	Эффективность					
	2 проц.	4 проц.	8 проц.	16 проц.	32 проц.	64 проц.
Кластер МВС-15000						
Вероятностный	0.91	0.88	0.83	0.72	0.42	0.39
Жизнь	0.95	0.92	0.90	0.83	0.47	0.19
Деление	0.94	0.90	0.92	0.78	0.59	0.48
Подв. гран. (г)	0.94	0.97	0.99	0.95	0.67	0.46
Подв. гран. (в)	0.94	0.94	1.03	0.69	0.48	0.28
Кластер МВС-1000						
Вероятностный	0.95	0.93	0.81	0.80	0.64	
Жизнь	1.03	0.98	0.97	0.98	0.80	
Деление	0.98	0.95	0.83	0.86	0.74	
Подв. гран. (г)	1.03	0.99	0.83	0.85	0.83	
Подв. гран. (в)	1.01	0.99	0.97	0.76	0.61	

Из табл. 1 следует, что эффективность всех динамических алгоритмов практически всегда выше вероятностного алгоритма. Это показывает, что даже при использовании высокоскоростных соединений (в частности Mirynet, 2 Гбит/с) большой обмен значительно снижает скорость вычислений. Это подтверждается специальным тестом, проведенным на 16 процессорах МВС-1000 для большого числа частиц на процессор (1.2 миллиона), в котором удалось выделить время работы процессоров, затраченное на различные операции. Результаты теста показаны в табл. 2. "Работа 1" — это время, за которое произведен перенос частиц, "Работа 2" — индексация (определение в какую ячейку попала частица), столкновение частиц и сбор макропараметров в ячейке, "Пересылка" — передача частиц между процессорами, "Сумма" — суммарное время вычислений.

Таблица 2

Время вычислений 500 шагов [сек]

Алгоритм	Работа 1	Работа 2	Пересылка	Сумма
Вероятностный	37.94	163.54	137.45	338.94
Жизнь	23.35	152.81	39.84	216.01
Деление	21.69	152.02	57.72	231.43
Подв. гран. (г)	23.12	153.93	41.56	218.62
Подв. гран. (в)	21.97	151.07	81.16	254.20

"Вероятностный" алгоритм выполняет "Работу 1" примерно в 1.6 раза медленнее: судя по всему, из-за копирования частиц в буфер для передачи (чтобы не передавать частицы по одной, а передать их все одним буфером, что снижает время на соединение). Это является алгоритмической частью переноса частиц и поэтому учитывается в "Работе 1". Таким образом, операция "память→память" замедлила работу процессора примерно в 1.6 раза.

Из табл. 1 также видно, что кластер МВС-15000, имеющий на 30% более производительные процессоры, с увеличением числа процессоров теряет эффективность гораздо быстрее, чем МВС-1000. Отчасти это связано с особенностями данного теста — про "недозагруженность" процессоров говорилось выше, но возможно, кроме того, существуют какие-то специфические, архитектурно-зависимые особенности.

Для оценки пригодности использования динамических алгоритмов при решении реальных больших задач, требующих более сотни процессоров и оперирующих миллионами ячеек и десятками миллионов частиц, были проведены расчеты обтекания модели спускаемой капсулы "Apollo", движущейся со скоростью 7600 м/с на высоте 75 км. Число Кнудсена для этого случая составляло 2.7×10^{-4}. В расчетах использовались 30 миллионов частиц в поле с 1.8 миллионами ячеек на 128 процессорах. Сравнивалась производительность вероятностного алгоритма, алгоритма с подвижными границами с начальной раз-

бивкой области вдоль потока и алгоритма "Жизнь". Для того чтобы изменяющееся поле течения на этапе установления не влияло на полученную эффективность расчетов, оценивалось время расчета 3000 шагов уже в стационарном течении.

Для решения этой задачи использовался кластер Московского межведомственного суперкомпьютерного центра, оснащенного вычислительными модулями с восемью процессорными ядрами "Xeon" в модуле. Соединение между модулями осуществлено на базе Infiniband со скоростью передачи 10 Гбит/с.

При вероятностной разбивке проведение трех тысяч шагов потребовало 2607 с, алгоритм с подвижными границами работал для этого 2502 с, а алгоритм "Жизнь" — 2986 с. Таким образом, один из адаптированных к течению алгоритмов показал ускорение на 4%, а другой — замедление на 13% по отношению к вероятностному алгоритму, при этом эффективность вероятностного алгоритма составила 0.66. В связи с тем, что такой расчет невозможно провести на однопроцессорном компьютере и стандартную формулу для вычисления эффективности применить нельзя, эффективность вычислялась по формуле $e = 1 - \dfrac{T_{||}}{T_M}$, где $T_{||}$ — время, потраченное на обеспечение взаимодействия между процессорами в многопроцессорном расчете. Поскольку в используемом программном коде весь межпроцессорный обмен (включая буферизацию) сосредоточен в одной процедуре, введение в этот код дополнительных счетчиков времени, измеряющих время выполнения этой процедуры, позволило достаточно корректно получить $T_{||}$.

Благодаря тому, что связь между вычислительными узлами осуществляется на скорости 10 Гбит/с, обмен данными становится несущественным фактором, и на первое место выходит требование равномерности загрузки процессоров. По этому критерию вероятностный алгоритм оказывается наиболее эффективным. Именно поэтому на этом суперкомпьютере эффективность алгоритма с вероятностной декомпозицией расчетной области равна и даже превышает эффективность адаптированных алгоритмов.

Таким образом, на суперкомпьютерах с очень быстрой связью предпочтительнее использовать вероятностный алгоритм — благодаря его простоте и универсальности. Однако при проведении расчетов на "домашних" кластерах использование предложенных методик вполне оправдано и может дать существенный выигрыш во времени расчета.

10. Заключение.

1. При параллелизации метода ПСМ объем пересылаемых данных может быть довольно большим; это обязательно должно учитываться при разделении области между процессорами, особенно при вычислениях на кластерах.

2. Критерием загрузки процессора в первом приближении может являться число частиц в ячейке. Объем пересылки удобнее оценивать количеством частиц, передаваемых между ячейками.

3. При разработке алгоритмов разделения области можно воспользоваться эмулятором параллельных вычислений, в котором используются ранее полученные поля течений. Критерием загрузки в этом эмуляторе является плотность потока, а пересылка оценивается по величине потока газа через грань ячейки.

4. Из рассмотренных алгоритмов разделения области наиболее удобными в применении являются алгоритмы "Жизнь" и "Деление на равные части". Эффективность алгоритма с "подвижными границами" сильно зависит от первоначального разделения области и поэтому не столь универсальна.

5. Эффективность алгоритмов разделения области зависит от архитектуры вычислительных кластеров и решаемой задачи. Поэтому при выполнении вычислений лучше попробовать несколько алгоритмов, а вычисление производить с помощью наиболее эффективного.

6. Расчеты на мощных суперкомпьютерах с очень быстрым сетевым интерфейсом показали, что значение использования динамических алгоритмов в этих условиях по сравнению с вероятностным алгоритмом не слишком высоко. Однако использование таких суперкомпьютеров доступно не всем. Проведение расчетов на более доступных, недорогих кластерах с сетевым интерфейсом до 1 Гбит/с можно значительно ускорить благодаря использованию адаптированных алгоритмов.

СПИСОК ЛИТЕРАТУРЫ

1. *Берд Г.* Молекулярная газовая динамика. М.: Мир, 1981.
2. *Bird G.A.* Molecular gas dynamics and the direct simulation of gas flows. Oxford: Clarendon Press, 1994.
3. *Ivanov M.S., Markelov G.N., Gimelshein S.F.* Statistical simulation of reactive rarefied flows: numerical approach and applications // AIAA Paper 98–2669.
4. *Ivanov M., Markelov G., Taylor S., Watts J.* Parallel DSMC strategies for 3D computations // Proc. Parallel CFD'96. P. Schiano, A. Ecer, J. Periaux, N. Satofuka (Eds.). Amsterdam: North Holland, 1997. 485–492.

5. *Antonov S., Pfreundt F.-J., Struckmeier J.* Adaptive load balance techniques in parallel rarefied gas simulations // J. of Computational Physics. 1997. **138**, Issue 2. 400–418.
6. *Nance R.P., Hassan H.A., Wilmoth R.G., Saltz J.* Parallel DSMC solution of three-dimensional flow over a finite flat plate // AIAA and ASME. Joint Thermophysics and Heat Transfer Conference, 1994. AIAA-1994-2019.
7. *Foster I.* Designing and building parallel programs: concepts and tool for parallel software engineering. Reading: Addison-Wesley Publishing Company, 1995.
8. *Kashkovsky A., Markelov G., Ivanov M.* An object-oriented software design for the direct simulation Monte Carlo method // AIAA 2001-2895.
9. *Kashkovsky A.V., Bondar Ye.A., Zhukova G.A., Ivanov M.S., Gimelshein S.F.* Object-oriented software design of real gas effects for the DSMC method // 24th Int. Symp. on Rarefied Gas Dynamics, Porto Giardino, Italy, July 10–16, 2004. AIP Conf. Proc. 2005. **762**, Issue 1. 583–588.

Поступила в редакцию
17.05.2009